Modular enumeration technology - a tool for creation of new algorithms family solving discrete programming problems

Vitaly O. Groppen*

North-Caucasian Institute of Mining and Metallurgy (State technological university),
44 Nikolajev str., Vladikavkaz, 362021, Russia

Abstract. The paper proposes a new enumeration technology based on modular enumeration ideology resulting in a new group of algorithms that numerically solve extremal problems with discrete variables. All descriptions of this approach are illustrated with examples. Presented are the conditions of application of the proposed technology, as well as analytical and experimental bounds of its effectiveness.

1 Introduction

All the efforts associated with the tendency to reduce the running time when solving various numerical problems can be identified as analytical and computer-based. The latter dominate either in relation to the use of parallel computing [1–5], or in relation to special procedures for a single processor, designed to solve extremal problems described by continuous as well as by discrete mathematical models, the search for the numerical values of multiple integrals, the roots of equations with n>1 variables, where the search for solutions in the general case relies on various enumeration procedures. As the implicit enumeration methods developed in the middle of the last century, such as dynamic programming, branch-and-bound (B&B) methods, backtracking [6–8] and their modifications [9–12] have several drawbacks:

1. the impossibility to predict a priori neither the number of iterations, nor its upper bound and, as a result, it is impossible to predict the gain in time from the use of these procedures as compared to the complete enumeration methods;
2. operating in a hostile environment, that is in the case when the B&B method, as well as backtracking, makes many mistakes when choosing the direction of search in the branch tree, there may be cases when the number of iterations by these methods exceeds the number of different complete plans, which means that running time of implicit enumeration methods sometimes exceeds the running time of the brute force method;
3. using dynamic programming the number of analyzed plans can also exceed that of different complete plans - it takes place when, searching a globally optimal solution, it is possible to cut off only a relatively few unpromising groups of variables' vectors due to the specifics of the being solved problem,

* Corresponding author:groppen@mail.ru

4. in [13] was proposed and developed the modular organization of the exhaustive search, which makes it possible to minimize repetitive computations at each iteration by storing their results in RAM and using them as needed.

The aim of his paper is in farther development of modular enumeration technology, it contains descriptions, analysis, and experimental verification of efficiency of the new realizations of the complete enumeration method including its modular organization related to the search for a globally optimal solution to extreme problems with Boolean variables. Examples and statistics below are based on knapsack problems solving [14, 15].

2 Basic ideas of modular enumeration technology for solving discrete programming problems with Boolean variables

The essence of the proposed approach is to implement two stages of solving any of the discrete programming problems: at the first stage preparation for the enumeration minimizing its computing volume is carried out, at the second stage the actual enumeration is carried out.

The first, preparatory stage of the modular enumeration procedure for finding globally optimal solutions to discrete programming problems consists of two steps.

At the first step of the first stage, all the variables are grouped into "m" modules and for each module all combinations of the values of the variables corresponding to this module are generated and stored. At the second step of the first stage for each combination of variables of each module, its' part of the objective function and constraints are calculated and stored.

The second stage also includes two steps:

A. Usage of the contents of the modules to generate all the complete plans, the corresponding values of the objective function and the left-hand parts of the constraints.
B. Their comparison as they are generated and thus choice of a vector of variables that satisfies all the restrictions with the best value of the objective function.

Below we assume that this approach is used for solving problems with Boolean variables of the form:

$$\begin{cases} F = \sum_{i=1}^{i=n} c_i z_i \to \max(\min); \\ \forall j : \sum_{i=1}^{i=n} b_{j,i} z_i \theta_j a_j; \\ \forall j : \theta_j \in \{\le;\ge;<;>;=\}; \\ \forall i : z_i = 1,0 \end{cases} \quad (1)$$

where $\boldsymbol{Z} = \{z_1, z_2, \ldots, z_n\}$ is a vector of Boolean variables, n – their number, whereas $\forall i, \forall j$: c_i, b_{ij}, and a_j are integer constants, $1 \le i \le n$; $1 \le j \le d$.

The optimal number of modules used for solving problem (1) meets the following problem:

$$\begin{cases} m \to \min; \\ \sum_{p=1}^{p=m} n_p = n; \\ k \cdot \sum_{p=1}^{p=m} (d+1) 2^{n_p} \le M, \end{cases} \quad (2)$$

where "m" is the number of modules ($m > 1$), n is the total number of variables of the problem (1); n_p is the number of variables, belonging to the p-th module, k is the proportionality coefficient, M is the amount of free RAM in the computer used.

It can be shown that the running time T_1 of the brute forcesearch for problem (1) globally optimal solution can be determined by the following expression:

$$T_1=(d+1)\cdot n\cdot 2^n\cdot t_0, \quad (3)$$

where t_0 is equal tothe sum of the times of addition and multiplication of two integer numbers, whereas the time of two numbers comparison is ignored.If the modular enumeration is used to solve problem (1), the running time of this algorithm does not exceed the T_2 value:

$$\begin{cases} T_2=(d+1)\cdot\lceil m\cdot 2^n + \sum_{p=1}^{p=m} n_p\, 2^{n_p}\rceil\cdot t_0 \rightarrow \min; & (4) \\ k\cdot\sum_{p=1}^{p=m} n_p \le M, & (5) \end{cases}$$

where "m" is the number of modules (m>1), n_p is the number of variables, belonging to the$p$-th module, k is the proportionality coefficient, M is the amount of free RAM in the computer used.

This poses three questions:

- What is the optimal number of modules for a specific combination "computer/problem"?
- What is the gain in running time for finding problem (1) globally optimal solution when applying modular enumeration if compared to the traditional enumeration scheme?
- What should be the optimal strategy of variables distribution between modules?

From (4) it follows that the minimum value of T_2 corresponds to the minimum value of m determined according to (5). Obviously, with a sufficiently large RAM size, the optimal value of m is equal to two. The latter makes it possible to determine the upper bound for the gain η in the time spent on the finding of a globally optimal solution to problem (1) when using modular enumeration in comparison with the traditional enumeration scheme. Considering that the minimum value of m is equal to two and ignoring the second term in square brackets in (4), the upper bound for the gain η in the running time - the ratio of the right-hand sides (3) and (4) results in the following equality:

$$\eta = T_1 / T_2 = n/m. \quad (6)$$

In other words, according to (6), the upper bound for the gain in the running timeη does not depend on the number of constraints d in (1) and is uniquely determined by the number of variables in this problem. Since when deriving (5), the first two mentioned above stages of searching for system (1) optimal solution by modular enumeration were ignored, associated with them time spent on generating and filling the modules was also ignored, that is why the value of η in (6) can be considered as the upper bound of the gain in computation time. Moreover, considering that, in the first approximation, the time spent by modular enumeration only on these two stages can be described as*:* $T_3= t_0\cdot n\cdot 2^{0.5n}$, the following expression is true:

$$\lim_{n\to\infty} T_3/T_2 = 0. \quad (7)$$

Since due to (4) - (5) system optimal number of modules in the modular enumeration algorithms for any amount of free RAM in the computer used can be determined, the problem arises of the optimal distribution of variables between the modules. This problem is solved by the following theorem:

Theorem 1. For the number of variables of the problem (1) "n", which is a multiple of the number of modules "m", the optimal is a uniform distribution of variables in these modules.

Proof: The running time T_1 corresponding to the search by modular enumeration algorithm with the uniform distribution of variables between the modules, i. e. $\forall 1<p\le n/m$*:* $n_p = n/m$ from (3) it follows:

$$T_1 = t_0\cdot(d-1)\cdot[n\cdot 2^{n/m} + (m-1)\cdot 2^n]. \quad (8)$$

We obtain a new, non-uniform distribution of variables between the modules, for which in the uniform distribution we remove g variables ($0 < g < n/m$) from one module and transfer them to the other module. The search time for problem (1) solution in this case is denoted as T_2:

$$T_2 = t_0\cdot(d-1)\cdot[\,[\,\frac{n}{m}(m-2)\cdot 2^{n/m} + (\frac{n}{m}-g)\cdot 2^{n/m-g} + (\frac{n}{m}+g)\cdot 2^{n/m+g} + (m-1)\cdot 2^n], \quad (9)$$

The difference T_2 - T_1 is designated bellow as ΔT. After expanding the parentheses and transformations, we get:

$$\Delta T = t_0 \cdot (d-1) \cdot 2^{n/m} \cdot [\,[(2^{-g} + 2^{g} - 2) + 2^{g} \cdot (2^{g} - 2^{-g})]. \quad (10)$$

It is easy to verify that for any integer values of g in the range [1 - n/m], the right-hand side of (10) is non-negative.

It follows that $T_1 < T_2$. The theorem 1 is proved.

3 Modular enumeration algorithms

Modular enumeration is a paradigm that allows you to generate a family of various enumeration algorithms. The four of these algorithms are presented and further explored below.

3.1 Algorithm 1

The main feature of Algorithm 1 first presented in [13], is in an organization of such full enumeration process, which reduces the running time by minimizing of repetitive computations.

Algorithm 1

Step 1. The entire set of variables $\boldsymbol{Z}$ is divided into $m = 2$ modules with the sets of variables $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$, for which the following conditions are true:

$$\begin{cases} \bigcup_{j=1}^{j=2} \boldsymbol{Z}_j = \boldsymbol{Z}; & (11) \\ \boldsymbol{Z}_1 \cap \boldsymbol{Z}_2 = \varnothing; & (12) \\ \forall i \in \{1, 2\}: |\,|\boldsymbol{Z}_i| - |\boldsymbol{Z}_{3-i}|\,| \le 1, & (13) \end{cases}$$

where $\boldsymbol{Z}_i$ is the subset of variables, corresponding to the i-th module (i= 1, 2).

Step 2. The values of components of the objective function as well as of constraints of problem (1) corresponding to each vector in each module, are calculated, and fixed in the RAM.

Step 3. A new vector of variables is generated by a new combination of in-RAM components of variables belonging to different modules. If there is no such a combination, then go to step 6.

Step 4. The new value of the goal function F is obtained as a sum of corresponding in-RAM components. If value F is "better" than the found earlier and fixed in RAM goal function value and all constraints of system (1) are satisfied, then go to the next step, otherwise, go to step 3.

Step 5. Value F and corresponding vector of variables are stored in RAM instead of kept earlier, go to step 3.

Step 6. The algorithm is completed. The values of F and of the vector of variables stored in RAM are problem (1) optimal solution.

Example 1. Using the above Algorithm 1, solve the knapsack problem [12] of the form:

$$\begin{cases} 7z_1 + 2z_2 + 4z_3 + 5z_4 \to \max; \\ 2z_1 + 4z_2 + 8z_3 + 3z_4 \le 12; \\ \forall i\text{: } z_i = 1,0. \end{cases} \quad (14)$$

Determine the effectiveness of the proposed approach in relation to problem (14), provided that m = 2.

1) $R = -\infty$.
2) Distribution of variables into two modules satisfying system (11) - (13):

$$\boldsymbol{Z}_1 = \{z_1; z_2\};\ \boldsymbol{Z}_2 = \{z_3; z_4\}.$$

Table 1. All the states of vectors of variables of each of the two modules.

m1				**m2**			
z1	z2	ΔF1	Δb1	z3	z4	ΔF2	Δb2
0	0	0	0	0	0	0	0
0	1	2	4	0	1	5	3
1	0	7	2	1	0	4	8
1	1	9	6	1	1	9	11

1) Generation of all states of the vectors of variables of each module and calculation of the corresponding components of the objective function ΔF_i and of the left side of the constraintof the system (14) Δb_i are presented in the Table 1 above.
2) Analysis of all the pairwise combinations of vectors of variables of both modules allows us to obtain a globally optimal problem (14) solution: $R = F_{max} = 14$, $\boldsymbol{Z}_{opt}$= {1,1,0,1}.
3) The effectiveness of the proposed approach in solving problem (14) is determined by the gain in running time of Algorithm 1 as compared to the brute force method solving the same problem. In the latter case, it can be shown that the running time of the search for a knapsack problem globally optimal solution is determined by the expression:

$$T_1 = n \cdot 2^{n+1} \cdot t_0, \tag{15}$$

where t_0 is the same, as used above in (2).

Keeping in mind that the number of modules is equal to two, the similar approach to estimating the search time for the same problem solution by modular enumeration allows us to determine this time as follows:

$$T_2 = t_0 \cdot \lceil n \cdot 2^{0.5n} + 2^{n+2} \rceil. \tag{16}$$

Ignoring the first term in square brackets in (16), the upper bound of the gain in the running time with the Algorithm 1 solving problem (14) is coinciding with the result of (8) – it's equal to η:

$$\eta = T_1/T_2 \approx 2. \tag{17}$$

The experimental results presented below in Section 4 also confirm the assessment of the Algorithm 1 efficiency given in (6).

3.2 Algorithm 2

Unlike Algorithm 1, Algorithm 2, being also a modular enumeration one, is organized in such a way that not only the number of arithmetic operations at each iteration, but also the number of iterations is minimized during the enumeration process. This is achieved in Algorithm 2 below by changing the order of generation of the vectors of variables at steps 8-12: during each iteration at step 12 a new vector of variables is generated, which consists of two parts corresponding to the best values of goal function components in each module. This strategy on the one hand allows us not to analyze vectors of variables, which correspond to a priori "bad" values of the objective function, thus giving us an opportunity to get a gain in Algorithm's 2 running time exceeding that which is determined by (8). On the other hand, due to the growth of time, spent for generation of each new vector of variables, the total running time of Algorithm 2 can also grow. Presented below description of Algorithm 2 as well as description of Algorithm 1 is based on usage of m=2 modules. Since the first two steps of Algorithms 1 and 2 are the same, the description of Algorithm 2 below begins with the third step.

Algorithm 2

Step 3. All vectors of variables of the first module are considered as unlabelled.

Step 4. Among the unlabeled variables' vectors of the first module, the one that corresponds to the best value of the components of objective function belonging to this

module, is selected. If all vectors of variables belonging to this module are labeled, then go to step 11.

Step 5. All vectors of variables of the second module are considered as unlabeled.

Step 6. Among the second module unlabeled vectors of variables, the one that corresponds to the best value of the components of objective function belonging to this module, is selected. If all vectors belonging to this module are already labeled, then go to step 4.

Step 7. The vectors of the variables selected in steps 4 and 6 of the last iteration are labeled and used for creation of a new vector of variables.

Step 8. If the value of the objective function, calculated with the values of the variables determined at the previous step, is "not better" than the previously found and stored in RAM value, then go to step 6, otherwise go to the next step.

Step 9. If the vector of variables created at step 7 of the last iteration satisfies all the conditions of problem (1), then go to step 10, otherwise, go to step 6.

Step 10. Replacing the value of the objective function stored in RAM with a new one corresponding to the vector of variables generated at step 7 of the last iteration and transit to step 4.

Step 11. The algorithm is completed. The best value of the objective function and corresponding vector of variables are stored in RAM.

It is easy to verify that, due to steps 4 and 6, the duration of each iteration of Algorithm 2 exceeds the similar parameter of Algorithm 1, however, due to steps 14 - 16, there is a good chance that the number of these iterations will decrease, and this possibility increase with the number of variables. The effectiveness of the above algorithm 2 is demonstrated below in two ways:

a. as applied to the solution of problem (14);
b. as shown by the statistical research presented in the next section.

Example 2. Using the above Algorithm 2, we solve the knapsack problem (14).

The first two steps of solving problem (14) by algorithms 1 and 2 coincide and lead to the construction of Table 1 (above), that is why they are not presented here. The sequence of problem (14) vectors of variables generated and analysed by Algorithm 2 is shown below in Table 2. Here F denotes the current value of the objective function, G denotes the current value of the left side of the inequality of problem (14), B is the right side of this inequality, and R is the maximum allowable value of the objective function.

Table 2. Vectors of variables analysed by Algorithm 2.

№	x_1	x_2	x_3	x_4	F	G	R	Remarks
1	1	1	1	1	18	17	-∞	$G>B$
2	1	1	0	1	14	9	14	$G<B$
3	1	0	1	1	16	13	14	$G>B$
4	1	0	0	1	12	5	14	$F<R$
5	0	1	1	1	11	15	14	$F<R$
6	0	0	1	1	9	11	14	$F<R$

The obtained result coincides with the previously found with Algorithm 1, but the difference lies in the fact that Algorithm 2 needed to analyze only 6 plans, whereas Algorithm 1 needed to analyze 16 plans. The latter does not mean that the search time for a solution to problem (14) by Algorithm 2 is less than that with Algorithm 1: as noted above, the duration of iterations of Algorithm 2 exceeds the duration of similar iterations of Algorithm 1.A detailed experimental analysis of the efficiency of the modular enumeration methods described above in relation to the knapsack problem is presented below in Section 4.

3.3 Algorithm 3

The third modification of the modular enumeration differs from the algorithms above: that is before the actual enumeration begins, the components of both modules are arranged in the order of deterioration of "their" objective function components. This approach simplifies the procedure of enumeration and, at the same time, gives us chance to reduce its running time by ignoring a priori "bad" vectors of variables. As the first two steps of Algorithms 1 and 3 coincide, the description of Algorithm 3 below begins with the third step.

Algorithm 3

Step 3. All the components of both modules are arranged in the order of deterioration of their components of goal function.
Step 4. i=1
Step 5. j=1.
Step 6. A new, not previously analysed vector of variables is obtained by combining its i-th component of the first module and the j-th component of the second module.
Step 7. Calculation of a new value of the objective function F, corresponding to the vector obtained at the previous step.
Step 8. If F is “better” than R, then go to the next step, otherwise go to step 16.
Step 9. If the constraints are met, then go to step 10, otherwise go to step 19.
Step 10. $R:=F$, the new vector of variables obtained at the 6-th step of the last iteration is stored in RAM, go to the next step .
Step 11. $i=i+1$.
Step 12. If $i>2^{|Z|}$, then go to step 15, otherwise go to step 5.
Step 13. $j=j+1$.
Step 14. If $j<2^{|Z\setminus Z|}$, then go to step 6, otherwise go to step 11.
Step 15. The algorithm is over. The best value of the objective function R and corresponding vector of variables are stored in RAM.

Example 3. Using the above Algorithm 3, solve the knapsack problem (14). As in the previous case the first two steps of solving problem (14) by algorithms 1 and 3 coincide and lead to the construction of Table 1 (above). During step 3 of Algorithm 3 Table 1 is transformed into Table 3 below:

Table 3. All the components of both modules are arranged in the order of deterioration of their goal function components.

m1				**m2**			
z_1	z_2	ΔF_1	Δb_1	z_3	z_4	ΔF_2	Δb_2
1	1	9	6	1	1	9	11
1	0	7	2	0	1	5	3
0	1	2	4	1	0	4	8
0	0	0	0	0	0	0	0

The sequence of problem (14) vectors of variables generated and analysed by Algorithm 3 is shown below in Table 4. In this case, the same designations as in Table 2 are used.

Table 4. Vectors of variables analysed by Algorithm 3.

№	x_1	x_2	x_3	x_4	***F***	***G***	***R***	**Remarks**
1	1	1	1	1	18	17	-∞	$G>B$
2	1	1	0	1	14	9	14	$G<B$
3	1	0	1	1	16	13	14	$G>B$
4	1	0	0	1	12	5	14	$F<R$
5	0	1	1	1	11	15	14	$F<R$

It is easy to verify that:

a) the volume of enumeration by Algorithm 3 is the smallest when solving (14) as compared to Algorithms 1 and 2;
b) this version of modular enumeration is close to the “meet in the middle” algorithm [15];
c) the results of solving (13) by all three algorithms above coincide.

3.4 Algorithm 4

The fourth modification of the modular enumeration differs from the algorithms above by the third step of the first stage, which is to discard the "unpromising" vectors of each module. The latter means that, if there are two vectors in any module, one of which has “better” values of parts of the objective function and constraints, than the other one, then the latter vector is considered as “unpromising”, and it can be removed. As a result, combining the rest contents of different modules during the second stage we get chance to analyze less than 2^ncomplete plans. It is easy to show that, if $\forall p,\ n_p = n/m$, the growth of running time as “payment” for shortening of the number of analyzed plans is proportional to the H value:

$$H=0.5\cdot m\cdot 2^{n/m}\cdot(2^{n/m}+1). \quad (18)$$

Step by step description of Algorithm 4 is presented below. As the first two steps of all the previous algorithms and algorithm 4 coincide, the description of this algorithm below begins with the third step.

Algorithm 4

Step 3. Analyzed is each pair of vectors in each module and, if there are two vectors in the same module, one of which has “better” values of parts of the objective function and constraints, than the other one, then the latter vector is removed.

Step 4. A new vector of variables is generated by new combination of in-RAM components of variables belonging to different modules. If there is no such a combination, then go to step 7.

Step 5. The new value of the goal function F is obtained as a sum of corresponding in-RAM components. If value$F$ is "better" than the found earlier and fixed in RAMgoal function value and all constraints of system (1) are satisfied,then go to the next step, otherwise, go to step 4.

Step 6. Value F and corresponding vector of variables are stored in RAM instead of kept earlier, go to step 4.

Step 7. The algorithm is completed. The values of F and of the vector of variables stored in RAM are problem (1) optimal solution.

Below is presented an example of using modular enumeration to solve a particular case of problem (1) - the knapsack problem, when solving problem (2) results in $m = 3$.

Example 4.

By the use of m=3 modules of modular enumeration solved is the following knapsack problem:

$$\begin{cases} F=2z_1+7z_2+4z_3+6z_4+3z_5+8z_6 \rightarrow max; \\ R=9z_1+3z_2+8z_3+2z_4+6z_5+4z_6 \le 10; \\ \forall i,\ z_i=1,0. \end{cases} \quad (19)$$

1. Results of the first two steps of the first stage are presented bellow in the tables 1÷3:

Table 5. All components of modules $m_1 \div m_3$.

m_1				m_2				m_3			
z_1	z_2	$F(m_1)$	$R(m_1)$	z_3	z_4	$F(m_2)$	$R(m_2)$	z_5	z_6	$F(m_3)$	$R(m_3)$
0	0	0	10	0	0	0	10	0	0	0	10
0	1	7	7	0	1	6	8	0	1	8	6
1	0	2	1	1	0	4	2	1	0	3	4
1	1	-∞	-2	1	1	10	0	1	1	11	0

2. After elimination of "unpromising" vectors of variables, Table 5 is transformed as follows:

Table 6. Modules $m_1 \div m_3$ after elimination of "unpromising" vectors.

m_1				m_2				m_3			
z_1	z_2	$F(m_1)$	$R(m_1)$	z_3	z_4	$F(m_2)$	$R(m_2)$	z_5	z_6	$F(m_3)$	$R(m_3)$
0	0	0	10	0	0	0	10	0	0	0	10
0	1	7	7	0	1	6	8	0	1	8	6
				1	1	10	0	1	1	11	0

It is easy to see that during the second stage after combining the rest vectors of different modules analyzed are only 18 of 64 complete plans. But this does not guarantee the gain in running time due to the duration of the third step of algorithm 4.

4 Experimental verification

The purpose of the experiments was to test the effectiveness of the above algorithms 1 - 4 in comparison with the brute force search in relation to the knapsack problem, the number of variables of which varied in the range from 3 to 20. Each algorithm's criterion of efficiency in relation to knapsack problems of a fixed dimension resulted in an average gain in a problem-solving time. The behaviour of the algorithms was analysed in relation to two types of environments: benevolent and hostile. Benevolent environment below corresponds to the case, when in (1) $\forall j$: a_j is close to the value $\sum_i b_{ij}$, whereas hostile environment is simulated by the opposite condition: $\forall j$: $a_j$is close to the $\min_i\{b_{ij}\}$.

The computer with the following parameters was used during our experiments:

Processor Celeron N4100 /J4125; RAM 8 Gb.; hard disk 1 Tb.

Operating system Windows 10 – 64.

The order of the experiments for the first three algorithms was determined by the following procedure:

4.1 Algorithm 5

Step 1. $n = 3$.

Step 2. Using Monte Carlo method we generated all the integer coefficients and constants of problem (1) for a given number of Boolean variables n in the range 1 - 10. Other parameters: $d = 1$, goal functions are maximized – generated are knapsack problems with the number of variables in the range 3 – 20.

Step 3. The problem obtained in the previous step is sequentially solved by software implementations of algorithms 1, 2, 3 and brute force. For each i-th algorithm (i=1, 2, 3) fixed are the corresponding running times $T_1(n)$, $T_2(n)$, $T_3(n)$, whereas $T(n)$ is the running time of the traditional organization of exhaustive search andthe gains in running time $\eta_1(n) = T(n)/T_1(n)$, $\eta_2(n) = T(n)/T_2(n)$, $\eta_3(n) = T(n)/T_3(n)$. Steps 2, 3 are repeated 10 times, after which the average values of the gain in time for each algorithm are recorded in the memory.

Step 4. $n = n + 1$.

Step 5. If $n < 21$, then go to step 2; otherwise, go to step 6.

Step 6. The algorithm is completed

Experimental dependences of average values of the gain in time as function of number of variables n for each algorithm -$\eta_1(n)$ (grey squares) for algorithm 1, $\eta_3(n)$ (black oblique crosses) for algorithm 3 and $\eta_2(n)$ (black triangles) for algorithm 2, are presented below in Figure 1 and Figure 2.

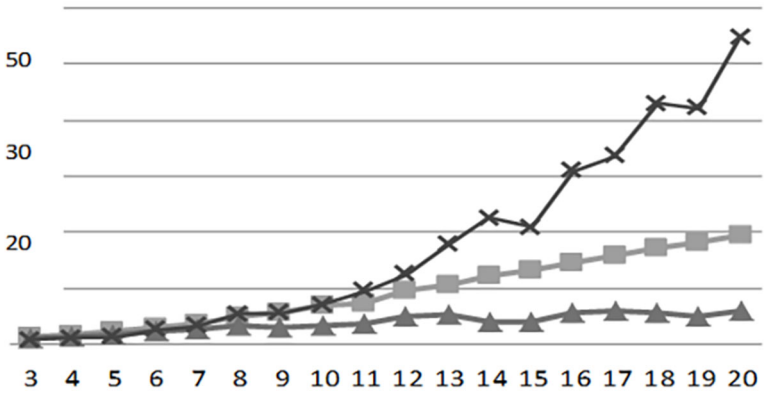

Fig. 1. The effectiveness of the above algorithms 1 - 3 in a hostile environment.

Procedure of experimental verification of Algorithm 4 differs from algorithm 5 in:

a) the number of variables n for each solved problem: $8 \le n \le 16$;

b) the number of problems w solved with coinciding values of n: $w=2$.

Experimental dependences of average values of the gain in running time as function of number of variables n for algorithm 4 - $\eta_4(n)$ (grey squares) and for algorithm 1 - $\eta_1(n)$ (black rhombuses) are presented in Figure 3.

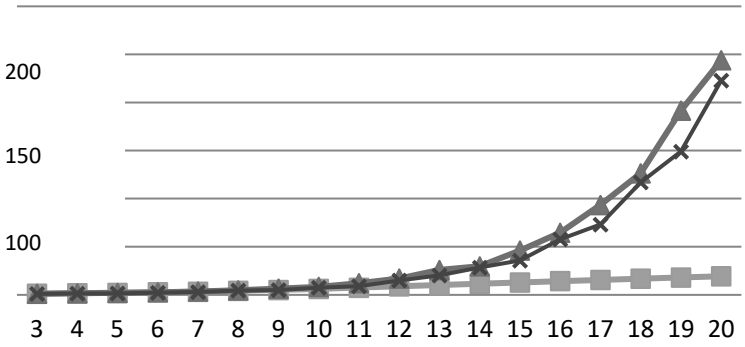

Fig. 2. The effectiveness of the above algorithms 1-3 in benevolent aria.

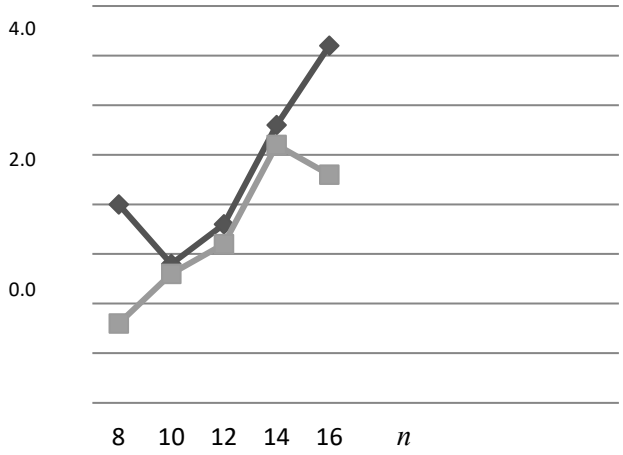

Fig. 3. The average values of the gain in time of the above algorithms 1 and 4.

Comparison of the average values of the gain in time of the first and fourth algorithms solving knapsack problems have shown that for n variables ($8 \le n \le 16$) more efficient was the first one.

5 Conclusion

In contrast to the methods of implicit enumeration, such as B&B, dynamic programming and backtracking, described above approach allows one to create algorithms permitting *a priori* to evaluate their effectiveness regardless of the specific numerical values of the coefficients and constants of problem (1).

The efficiency of these algorithms depends on the environment in which they operate, but their common feature is the increase in efficiency with an increase in the number of variables of the problem being solved.

It is easy to make sure that in the case when the number of modules m is equal to two, the third modification of modular enumeration is close to the "meet in the middle" algorithm [15].

The gain in the running time presented by (5) is true only for the algorithm 1 above.

Algorithm 1 have shown smaller average running time solving knapsack problems with n variables ($8 \le n \le 16$) than more complicated algorithm 4.

References

1. R.L. Boehning, R.M. Butler, B.E. Gillett, European Journal of Operational Research **34**, 393-398 (1988).
2. L. Brochard, Parallel Computing **12(1)**, 21-44 (1989)
3. M.J. Diamond, C. Kimbel, C.L. Rennolet, S.E. Ross, PICO: parallel implementation of combinatorial optimization, *Workshop on Parallel Computing of Discrete Optimization Problems* (Univ. Minneapolis-AHPC, 1991)
4. B.W. Wah, G.J. Li, C.F. Yu, Computer **18**, 93-108 (1985)
5. J. Casti, M. Richardson, R. Larson, JOTA **12(4)**, 423-438 (1973)
6. A.H. Land, A.G. Doig, Econometrica **28(3)**, 497-520 (1960)

7. C.A. Brown, P.W. Purdom, IEEE PAMI, 309-315 (1982)
8. R. Bellman, The Theory of Dynamic Programming. The RAND Corporation (1954)
9. Panos M. Pardalos et al., Combinatorial and Global Optimization, World Scientific Book, 335 (2002)
10. G. Desaulniers, J. Desrosiers, S. Spoorendonk, Cutting Planes for Branch-and-Price Algorithms. Published online 29 October 2011 in Wiley Online Library (wileyonlinelibrary.com). Wiley Periodicals, Inc. (2011)
11. D.R. Morrison, New methods for branch-and-bound algorithms. *Dissertation submitted for the degree of Doctor of Philosophy in Computer Science* in the Graduate College of the University of Illinois at Urbana-Champaign (2014)
12. V.O. Groppen, *Analysis of the Effectiveness of Composite Versions of Backtracking Algorithms*. Conference proceedings, RusAutoConf 2020: Available: Advances in Automation II, (Springer, 2021), pp. 235-244
13. V.O. Groppen, *A New Method Searching Globally Optimal Solutions to Discrete Optimization Problems*. Proceedings of the Future Technologies Conference (FTC) **3**, 486-494 (2021)
14. L. Caccetta, A. Kulanoot, Nonlinear Analysis **47**, 5547-5558 (2021). https://www.doi.org/10.1016/s0362-546x(01)00658-7
15. H. Ellis, S. Sartaj, Journal of the Association for Computing Machinery **21(2)**, 277-292 (1974)